**Лабораторна робота №1**

**Тема: Особливості виконання операцій та операторів у мові JavaScript. Типи даних. Приведення типів.**

***Мета роботи:*** вивчити відмінності у роботі операцій та операторів у мові JavaScript від інших мов; ознайомитись з базовими типами даних; вивчити особливості приведення типів.

**1 Основні теоретичні відомості**

### 1.2 Змінні

Змінна складається з імені та виділеної області пам'яті, яка їй відповідає.

Для оголошення або, іншими словами, створення змінної використовується ключові слова var (застаріло) let або const (для оголошення констант):

let message;

Після оголошення, можна записати в змінну дані:

let message;

message = 'Hello'; // збережемо в змінну рядок

Ці дані будуть збережені у відповідній області пам'яті і надалі доступні при зверненні до змінної. Для стислості можна поєднати оголошення змінної і запис даних:

let message = 'Hello!';

Можна навіть оголосити кілька змінних відразу:

let user = 'John', age = 25, message = 'Hello';

Можна оголосити дві змінні і копіювати дані з однієї в іншу:

let hello = 'Hello world!';

let message;

// скопіювали значення

message = hello;

alert (hello); // Hello world!

alert (message); // Hello world!

У оголошень змінної через let є три основні відмінності від var:

1. Область видимості змінної let – блок {...}.

Змінну, оголошену через var, видно всюди в функції. Змінну, оголошену через let, видно тільки в рамках блоку {...}, в якому оголошено. Це, зокрема, впливає на оголошення всередині if, while або for.

2. Змінну let видно тільки після оголошення.

Змінні var існують і до оголошення, просто вони рівні undefined.

alert (a); // undefined

var a = 5;

Зі змінними let все простіше. До оголошення їх взагалі неіснує.

3. При використанні в циклі, для кожної ітерації створюється своя змінна.

Змінна var – одна на всі ітерації циклу і її видно навіть після циклу. З змінної let – все по-іншому. Кожному повторення циклу відповідає своя незалежна змінна let. Якщо всередині циклу є вкладені оголошення функцій, то в замиканні кожної буде та змінна, яка була при відповідній ітерації.

**Імена змінних**

На ім'я змінної в JavaScript накладено всього два обмеження.

* ім'я може складатися з: букв, цифр, символів $ і \_
* перший символ не повинен бути цифрою.

let myName;

let test123;

let $ = 1; // змінна з ім'ям '$'

let \_ = 2; // змінна з ім'ям '\_'

alert ($ + \_); // 3

А такі змінні були б неправильними:

let 1a; // початок не може бути цифрою

let my-name; // дефіс '-' не є дозволеним символом

Регістр букв має значення. Змінні apple і AppLE – дві різні змінні.

Застосування кирилиці допустиме, але не рекомендуються

**Зарезервовані імена**

Існує список зарезервованих слів, які не можна використовувати для змінних, так як вони використовуються самою мовою, наприклад: var, class, return, export і ін.

### 1.2 Типи даних

Як і будь-яка інша мова програмування, JavaScript підтримує вбудовані структури і типи даних. В JavaScript існує кілька основних типів даних.

**Число «number»**

Єдиний тип число використовується як для цілих, так і для дрібних чисел.

let n = 123;

n = 12.345;

Існують спеціальні числові значення Infinity (нескінченність) і NaN (помилка обчислень).

Помилка обчислень NaN буде результатом некоректної математичної операції, наприклад:

Ці значення формально належать типу «число», хоча, звичайно, числами в їх звичайному розумінні не є.

**Рядок «string»**

let str = "Hello world";

str = 'Одинарні лапки теж підійдуть';

В JavaScript одинарні та подвійні лапки рівноправні. Можна використовувати або ті або інші.

**Булевий (логічний) тип «boolean»**

У нього всього два значення: true (істина) і false (брехня).

Як правило, такий тип використовується для зберігання значення типу так / ні, наприклад:

let checked = true; // поле форми позначено галочкою

checked = false; // поле форми не містить галочки

**Спеціальне значення «null»**

Значення null не відноситься ні до одного з типів вище, а утворює свій окремий тип, що складається з одного значення null:

let age = null;

В JavaScript null не є «посиланням на неіснуючий об'єкт» або «нульовим покажчиком», як в деяких інших мовах. Це просто спеціальне значення, яке має сенс «нічого» або «значення невідомо».

**Спеціальне значення «undefined»**

Значення undefined, як і null, утворює свій власний тип, що складається з одного цього значення. Воно має сенс «значення не присвоїли».

Якщо змінна оголошена, але в неї нічого не записано, то її значення як раз і є undefined:

let x;

alert (x); // виведе "undefined"

Можна привласнити undefined і в явному вигляді, хоча це робиться рідко:

В явному вигляді undefined зазвичай не присвоюють, так як це суперечить його змістом. Для запису в змінну «порожнього» або «невідомого» значення використовується null.

**Об'єкти «object»**

Перші 5 типів називають "примітивними". Окремо стоїть шостий тип: «об'єкти». Він використовується для колекцій даних і для оголошення більш складних сутностей.

Оголошуються об'єкти за допомогою фігурних дужок {...}, наприклад:

const user = {name: "Jhon"};

**Оператор typeof**

Оператор typeof повертає тип аргументу. У нього є два синтаксису: без дужок і з дужками. Працюють вони однаково, але перший синтаксис коротше.

typeof x; // синтаксис оператора

typeof (x); // синтаксис функції:

Результатом typeof є рядок, що містить тип:

typeof undefined // "undefined"

typeof 0 // "number"

typeof true // "boolean"

typeof "foo" // "string"

typeof {} // "object"

typeof null // "object"[[1]](#footnote-2)

typeof function () {} // "function"[[2]](#footnote-3)

### 1.3 Арифметичні оператори

Для роботи зі змінними, зі значеннями, JavaScript підтримує всі стандартні оператори, більшість яких є і в інших мовах програмування.

У операторів є своя термінологія, яка використовується в усіх мовах програмування:

– Операнд – те, до чого застосовується оператор. Наприклад: 5 \* 2 - оператор множення з лівим і правим операндами. Інша назва: «аргумент оператора».

– унарним називається оператор, який застосовується до одного виразу. Наприклад, оператор унарний мінус "–" змінює знак числа на протилежний:

let x = 1;

x = -x;

alert (x); // -1, застосували унарний мінус

– Бінарним називається оператор, який застосовується до двох операндів. Той же мінус існує і в бінарній формі:

let x = 1, y = 3;

alert (y - x); // 2, бінарний мінус

**Додавання рядків, бінарний +**

Зазвичай за допомогою плюса '+' складають числа. Але якщо бінарний оператор '+' застосувати до рядків, то він їх об'єднує в один:

let a = "новий" + "рядок";

alert (a); // новийрядок

Інакше кажуть, що «плюс здійснює конкатенацію (додавання) рядків». Якщо хоча б один аргумент є рядком, то другий буде також перетворений до рядка. Причому не важливо, праворуч або ліворуч знаходиться операнд-рядок, в будь-якому випадку нерядковий аргумент буде перетворений до рядка. наприклад:

Це приведення до рядка – особливість виключно бінарного оператора "+".

Решта арифметичні оператори працюють тільки з числами і завжди приводять аргументи до числа.

**Приведення до числа, унарний плюс +**

Унарний плюс нічого не робить з числами. Проте, він широко застосовується, так як його «побічний ефект» - перетворення значення в число. Наприклад, коли отримуємо значення з HTML-полів або від користувача, то вони зазвичай у формі рядків.

А що, якщо їх потрібно, наприклад, скласти? Бінарний плюс складе їх як рядки.

let apples = "2";

let oranges = "3";

alert (apples + oranges); // "23",

Тому використовуємо унарний плюс, щоб ппривести значення операнда до числа:

alert (+apples + +oranges); // 5,

Спочатку виконаються унарні плюси, які приведуть рядки до чисел, а потім - бінарний '+' їх складе.

**Пріоритет**

У тому випадку, якщо у виразі є декілька операторів – порядок їх виконання визначається пріоритетом. В JavaScript існує ціла таблиця пріоритетів, в якій кожному оператору заданий числовий пріоритет. Той, у якого число більше – виконається раніше. Якщо пріоритет однаковий, то порядок виконання – зліва направо.

**Присвоєння**

В таблиці пріоритетів також є оператор присвоювання =. У нього – один з найнижчих пріоритетів: 3. Саме тому, коли змінній щось привласнюють, наприклад, x = 2 \* 2 + 1 спочатку виконається арифметика, а вже потім – відбудеться присвоювання =.

let x = 2 \* 2 + 1; // 5

Можливо присвоювання по ланцюжку:

let a, b, c;

a = b = c = 2 + 2;

alert (a); // 4

alert (b); // 4

alert (c); // 4

Таке присвоювання працює праворуч-ліворуч, тобто спочатку обчислити саме праве вираз 2 + 2, присвоїти в c, потім виконається b = c і, нарешті, a = b.

Оператор "=" повертає значення. Завдяки цьому присвоювання можна використовувати як частину більш складного вираження:

let a = 1;

let b = 2;

let c = 3 - (a = b + 1);

alert (a); // 3

alert (c); // 0

**Взяття залишку %**

Дозволяє знайти залишок від ділення одного цілого числа на інше ціле число.

alert (5 % 2); // 1, залишок від ділення 5 на 2

alert (8 % 3); // 2, залишок від ділення 8 на 3

alert (6 % 3); // 0, залишок від ділення 6 на 3

**Інкремент / декремент: ++, --**

Однією з найбільш частих операцій в JavaScript, як і в багатьох інших мовах програмування, є збільшення або зменшення змінної на одиницю.

Для цього існують навіть спеціальні оператори[[3]](#footnote-4):

Інкремент ++ збільшує на 1:

let i = 2;

i++; // більш короткий запис для i = i + 1, i = 2;

Декремент -- зменшує на 1:

let i = 2;

i--; // більш короткий запис для i = i – 1, i = 1;

Викликати ці оператори можна не тільки після, але і перед змінною: i++ (називається «Постфіксна форма») або ++i ( «префіксна форма»).

Обидві ці форми записи роблять одне і те ж: збільшують на 1.

Проте, між ними існує різниця. Її видно тільки в тому випадку, коли хочемо не тільки збільшити / зменшити змінну, але і використовувати результат в тому ж виразі.

let i = 1;

let a = ++i; // a = 2;

Постфіксна форма i++ відрізняється від префіксної ++i тим, що повертає старе значення, яку було до збільшення.

let i = 1;

let a = i ++; // a = 1;

Якщо результат оператора не використовується, а потрібно тільки збільшити / зменшити змінну – без різниці, яку форму використовувати.

Якщо треба тут же використовувати результат, то потрібна префіксна форма. Якщо потрібно збільшити, але треба значення змінної до збільшення – постфіксна форма.

Інкремент / декремент можна використовувати в будь-яких виразах. При цьому він має більш високий пріоритет і виконується раніше, ніж арифметичні операції:

let i = 1;

alert (2 \* ++i); // 4

let j = 1;

alert (2 \* j++); // 2, виконався раніше але значення повернув старе

**Побітові оператори**

Побітові оператори розглядають аргументи як 32-розрядні цілі числа і працюють на рівні їх внутрішнього двійкового представлення.

Підтримуються наступні побітові оператори:

* AND (і) (&)
* OR (або) (|)
* XOR (побітове виключаюче або) (^)
* NOT (заперечення) (~)
* LEFT SHIFT (лівий зсув) (<<)
* RIGHT SHIFT (правий зсув) (>>)
* ZERO-FILL RIGHT SHIFT (правий зсув із заповненням нулями) (>>>)

Вони використовуються досить рідко.

**Скорочена арифметика з привласненням**

Часто потрібно застосувати оператор до змінної і зберегти результат в ній же, наприклад:

let n = 2;

n = n + 5;

n = n \* 2;

Цей запис можна вкоротити за допомогою суміщених операторів, ось так:

let n = 2;

n + = 5; // тепер n = 7 (працює як n = n + 5)

n \* = 2; // тепер n = 14 (працює як n = n \* 2)

Так можна зробити для операторів +, -, \*, /,% і бінарних <<, >>, >>>, &, |, ^.

Виклик з привласненням має в точності такий же пріоритет, як звичайне присвоювання, тобто виконається після більшості інших операцій:

let n = 2;

n \* = 3 + 5;

alert (n); // 16 (n = 2 \* 8)

**2 Завдання до роботи**

1. Створіть скрипт lab1-1.js, у якому:

– реалізуйте функцію isInteger(num), яка повертає true, якщо num – ціле число, інакше – false. Приклад використання:

console.log( isInteger(1) ); // true

console.log ( isInteger(1.5) ); // false

console.log ( isInteger(-0.5) ); // false

– реалізуйте функцію findPrimes(a, b), яка приймає числа *a* і *b*, і знаходить усі прості числа, які знаходяться на інтервалі [*a*, *b*]. Функція повинна результат виводити у консоль.

Створіть файл lab1-1.html, підключіть створений скрипт. Розмістіть тег <script></script>, у якому викличте написані функції з метою демонстрації правильності роботи написаного коду.

2. Ознайомитись з реалізацією регулярних виразів у мові JavaScript:

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Regular_Expressions>

Створіть скрипт lab2-2.js, у якому реалізуйте наступні функції:

|  |  |
| --- | --- |
| isIPAddress(ip) | Перевіряє, чи є вказаний рядок ІР-адресою (повертає true або false) |
| findRGBA(text) | Знаходить у заданому тексті рядок формату  rgba(r, g, b, a)  що відповідає кольору у форматі RGBA і повертає його. Якщо такого рядка не знайдено, то повернути null. |
| findHexColor(text) | Знаходить у заданому тексті колір у форматі #ABC або #ABCDEF. Якщо не знайдено, то повернути null. |
| findTags(text, tag) | Знаходить у заданому тексті усі теги із заданим ім’ям. Приклад використання:  let tags = findTags(s, "img"); |
| findPosNum(text) | Знаходить у тексті усі додатні числа, і повертає їх у вигляді масиву |
| findDates(text) | Знаходить у тексті усі дати записані у форматі  РРРР-ММ-ДД |
| findEmail(text) | Знаходить всі слова в тексті, які є коректними email адресами і повертає їх у вигляді масиву |

Створіть файл lab1-2.html, підключіть створений скрипт. Розмістіть тег <script></script>, у якому викличте написані функції з метою демонстрації правильності роботи написаного коду.

3. Створіть репозиторій JSLab1 на <https://github.com/>. Закомітьте у нього виконані завдання та надайте доступ користувачу zikratiy@gmail.com

1. це офіційно визнана помилка в мові, яка зберігається для сумісності. Насправді null - це не об'єкт, а окремий тип даних [↑](#footnote-ref-2)
2. функції не є окремим базовим типом в JavaScript, а підвидом об'єктів [↑](#footnote-ref-3)
3. Інкремент / декремент можна застосувати тільки до змінної. Код 5++ дасть помилку. [↑](#footnote-ref-4)